Dependable Transaction for Electronic Commerce

Hao Wang[1], Heqing Guo[1], Manshan Lin[1], Jianfei Yin[1], Qi He[2], and Jun Zhang[2]

[1] School of Computer Science & Engineering, South China University of Technology, Guangzhou, China 510640
guozhou@scut.edu.cn
{iswanghao, lmshill, yjhhome}@hotmail.com
[2] Computer Engineering School, Nanyang Technological University, Singapore 639798
{qihe0001, jzhang}@ntu.edu.sg

Abstract. Electronic transaction becomes common practice in real world business. This paper focuses on the issue of dependability in critical transactions like electronic payment, electronic contract signing. Recent fair protocols can recover transactions from network crashes, but cannot survive local system crashes. A two-party dependable transaction protocol is proposed. During the protocol, both parties can recover the transaction from network and local system failures in a transparent way, which means that after the recovery, outcome messages would be just the same as those from a successful run of the transaction.

1 Introduction

Electronic transaction becomes common practice in real world business. When the transaction between organizations is executed on network, they may face risks of broken fairness in case of network failures, local systems failures [3], cheating behavior of either involved organization, and so on. So it is very important for them to follow some kind of transaction protocol assuring *dependability*. Dependability assures fairness for involved parties and recoverability from failures. Fairness means that when the electronic transaction terminates, either both parties get their expected items, or neither does. A Trusted Third Party (TTP) is involved as Pagnia and Garner [6] have proved that no definite fairness can be achieved without a TTP.

We first set up the application scenario for our transaction protocol: company B (the client, denoted as Bob) is going to buy some electronic goods from company A (the merchant, denoted as Alice) and they have settled on the goods and the price. Now they need to finish the exchange of Bob's check with Alice's goods. Bob's check is composed of his bank-certified account information, goods information and can be validated only after signed by his signature. With that signed check, Alice can get her money paid from Bob's bank.

1.1 Related Work

1.1.1 Fair Protocol Capable of Recovery from Network Crashes

In 1996, Asokan et al. [1] introduces the idea of optimistic approach and presents fair protocols with offline TTP, in which TTP intervenes only when an error occurs (net-

O. Gervasi et al. (Eds.): ICCSA 2005, LNCS 3482, pp. 691 – 700, 2005.

work error or malicious party's cheating). But the recovered messages are different from those produced by the sender or the recipient, which make the protocols suffer from bad publicity and weak fairness, as the recovered messages may lose some functionalities of the original ones. *Invisible TTP* is first introduced by Micali [5] to solve this problem. The TTP can generate exactly the same evidences as the sender or the recipient. In this way, judging the outcome evidences and received items cannot decide whether the TTP has been involved, so that the recovery is done in a transparent way.

Using *convertible signatures* (CS) is the recently focused approach to realize transparent recovery. It means to firstly send a partial committed signature that can be converted into a full signature (that is a normal signature) by both the TTP and the signer. Recently, Park et al. [7] present a very efficient protocol in which the output evidences are standard RSA signatures and the partial signature is non-interactively verifiable. But very soon, Dodis and Reyzin [2] break the scheme by proving the TTP can obtain Alice's entire secret key with only her registration information. In the same paper, they propose a new CS scheme (*DR signature scheme*) to produce an equally efficient but more secure protocol.

But all these protocols have not considered cases of systems crashes and assumed that local systems of Alice, Bob and TTP are all stable.

1.1.2 Recovery Methods for Local Systems Crashes

Liu et al. [3] have proposed the *Semantics-based Message Logging* (SbML method) to enable recovery of local systems crashes. The SbML is a logging method balanced between pessimistic logging (log all messages before sending out) and optimistic logging (message processing and logging is separated). Involved parties can define their critical points (called *point-of-no-return*) in the protocol run and message will be logged before they enter the defined points.

This logging method works in protocols with online TTP. But when it comes to offline TTP and invisible TTP, fairness after crashes can be potentially broken. Cases of broken fairness are as following:

Case 1.1: after Alice sends out the first message, her system crashes; when Bob get the message, he can invoke the recover sub-protocol to get the final expected messages; if Alice fails to recover her system before TTP's recovered messages arrive, her fairness will be broken. So simply using their logging method is not enough to guarantee fairness.

Case 1.2: the offline TTP has not logged the variables: *recovered* and *aborted*, if TTP crashes after a successful abort operation requested by Alice; at this time, Alice has quitted the transaction since her request has been confirmed; but if Bob submit a recover request after TTP recovers, TTP will recover the transaction and send proper recovered messages to Alice and Bob; in this case, the message cannot arrive Alice, so fairness for Alice is broken.

1.2 Our Work

In this paper we first define the property of *Dependability* of transaction protocol. Then we present a transaction protocol based on DR signature scheme. To enable transparent recovery of crashes of network and local systems, we adapt the Seman-

tics-based Message Logging method and introduce a new inquiry sub-protocol. Finally we prove that the transaction protocol is dependable.

The remainder of the paper is structured as follows. In Section 2, we define the dependability property of electronic transactions. Section 3 presents the transaction protocol in payment scenario. Section 4 analyzes the protocol in details. Some concluding remarks are given in Section 5.

2 Dependability of Electronic Transactions

Markowitch et al.[4] study many former fairness definitions and present a well-knitted definition. Recently, Wang and Guo [20] present a set of new requirements for fair protocols with invisible TTP. Based on that, we extract 5 properties of transaction protocols and we say a protocol is dependable if it satisfies all these properties.

Definition 1. Effectiveness
A transaction protocol is *effective* if there exists a successful exchange of both parties' expected items.

Definition 2. Fairness
A transaction protocol is *fair* if when the protocol run ends, either both parties get their expected items or neither of them gets anything useful.

Definition 3. Timeliness
A transaction protocol is *timely* if the protocol can be completed in a finite amount of time while preserving fairness for both exchangers.

Definition 4. Non-repudiability
A transaction protocol is *non-repudiable* if when the exchange succeeds, either payer or payee cannot deny (partially or totally) his/her participation.

Definition 5. Transparent recoverability
A transaction protocol is *transparent recoverable* if after a successful exchange, the result evidences of origin/receipt and exchanged items are indistinguishable in respect to whether TTP has been involved.

With all these properties' definitions, we can define the dependability as following:

Definition 6. Dependability
A transaction protocol is *dependable* if it assures effectiveness, fairness, timeliness, non-repudiability and transparent recoverability.

3 A Dependable Payment Protocol with Transparent Recovery

In this section, we present a dependable protocol in the payment scenario described in Section 1. The protocol uses the DR signature as an important cryptographic tool. So we first briefly describe this signature scheme. Then with assumptions clearly presented, all the five parts of the protocol is described in details.

3.1 Dodis-Reyzin Convertible Signature Scheme

The DR signature is based on a recent widely-used RSA-like signature scheme called *gap Diffie-Hellman (GDH) signature* and the corresponding *GDH groups* (see [2] section 4 for detailed description).

GDH Signature. Assume G is a multiplicative group of prime order p. Key generation algorithm of the GDH signature scheme picks a GDH group of order p, and random $g \in G, x \in Z_p$. It computes $h = g^x$, and set the public key to be (g, h) (G, p is public accessible), and the secret key to be x. To sign a message m, one computes $\sigma = H(m)^x$, where $H(m)$ is a random oracle. To verify σ, one outputs $V_{DDH}(g, h, H(m), \sigma)$, that is, test if $\log_g h = \log_{H(m)} \sigma$ (outputting 1 means being equal). One can easily find that a secure zero-knowledge proof can accomplish this test.

DR Signature. This CS signature scheme contains one register procedure and several signing/verifying algorithms.

Register Procedure. Signer (say Alice) chooses random $g \in G, x, x_1 \in Z_p$, computes $x_2 = x - x_1 \bmod p, h = g^x, h_1 = g^{x_1}$, and sets her public key $pk = (g, h)$, secret key $sk = (x, x_1)$, partial public key $ppk = h_1$, partial secret key $psk = x_2$, then she sends the pk, ppk, psk to the TTP, the TTP will check whether $h = h_1 g^{x_2}$ so that it can finish the signature conversion.

Signing/Verifying Algorithms of Full Signature. They are just the signing/verifying algorithms of normal GDH signature: $FS(m) = \sigma = H(m)^x$, $Ver(m, \sigma) = V_{DDH}(g, h, H(m), \sigma)$.

Signing/Verifying Algorithms of Partial Signature. Similar with former ones but using the public key h_1: $PS(m) = \sigma' = H(m)^{x_1}$, $PVer(m, \sigma') = V_{DDH}(g, h_1, H(m), \sigma')$.

Converting Algorithm. The TTP run this algorithm $Convert(m, \sigma')$ to convert $PS(m)$ to $FS(m)$: it will first check whether $PVer(m, \sigma') = 1$, if holds, it outputs $FS(m) = \sigma' H(m)^{x_2}$.

Dodis and Reyzin have proved the DR signature scheme is just as secure as the normal GDH signature scheme ([2] Theorem 3).

3.2 The Protocol

Based on the application scenario set in Section 1, we first state our protocol's assumptions as following:

Communication Network. We assume the communication channel between Alice and Bob is unreliable and channels between exchangers (Alice/Bob) and TTP are resilient. Messages in a resilient channel can be delayed but will eventually arrive. On the contrary, messages in unreliable network may be lost.

Cryptographic Tools. Encryption tools including symmetric encryption, asymmetric encryption and normal signature is secure. In addition, the adopted signature scheme is message recovery.

Honest TTP. The TTP should send a valid and honest reply to every request. Honest means that when the TTP is involved, if a recover decision is made, Alice gets the payment and Bob gets the goods; if a abort decision is made, Alice and Bob get the abort confirmation and they cannot recover the exchange in any future time.

Local Systems. Local systems of Alice, Bob and TTP are recoverable with proper message logging including logging before *point-of-no-return* [3].

To describe the protocol, we need to use several notations concerning the necessary cryptographic tools:

- $E_k()/D_k()$: a symmetric-key encryption/decryption function under key k
- $E_X()/D_X()$: a public-key encryption/decryption function under pk_X
- $S_X()$: ordinary signature function of X
- k: the key used to cipher goods
- pk_X/sk_X: public/secret key of X
- $cipher = E_k(goods)$: the cipher of goods under k
- $X \rightarrow Y$: transmission from entity X to Y
- $h()$: a collision resistant one-way hash function
- *goods*: goods destined to B
- *check*: the check destined for A, it contains transaction identity, goods identity, price information, B's account information, etc
- l: a label that uniquely identifies a protocol run
- f: a flag indicating the purpose of a message

Registration Sub-protocol. To participate in a payment protocol, both Alice and Bob need to run the register procedure with the TTP as required by DR signature. Note that it will not affect the security if they share a same g. Bob also need to send his check for the TTP to verify its validity.

Main Protocol. After Alice and Bob settle the price and the goods, they can follow the main protocol. Note that they both make their own messages logged on stable storage before run the protocol:

Step 1, Alice sends encrypted goods (*cipher*) with the key k encrypted by the TTP's public key ($E_{TTP}(k)$), her partial signature on them (a=*cipher*, $E_{TTP}(k)$, $PS_A(a)= \sigma_A'$) to initiate the payment process.

Step 2, if Bob decides to give up or he doesn't receive Alice's message in time, he can simply quit and retain fairness. When he receives the message, he will first run $PVer(a, \sigma_A')$, if it equals 1, he will send his *check* and his partial signature on it ($PS_B(check)= \sigma_B'$) to Alice. Otherwise, he quits the protocol.

Step 3, if Alice decides to give up or she doesn't receive Bob's message in time, she can invoke the *abort* sub-protocol to prevent a later resolution by the TTP. When she receive the message, she will first run $PVer(check, \sigma_B')$, if it equals 1, she will log the message and the state information, then send k and her full signature on a ($FS_A(a)= \sigma_A$) to Bob. Otherwise, she also invokes the *abort* sub-protocol.

Step 4, if Bob detects that his channel with Alice is broken or doesn't receive the message in time, he can invoke the *recover* sub-protocol. When he receive the mes-

sage, he will check whether k can decrypt the *cipher* and the *goods* is satisfactory, also he will run $Ver(a, \sigma_A)$, if all these checking pass, he will log the message and the state information, then send his *check* and his full signature on it ($FS_A(check) = \sigma_B$) to Alice. Otherwise, he will invoke the *recover* sub-protocol.

Step 5, if Alice detects that her channel with Bob is broken or doesn't receive the message in time, she can invoke the *recover* sub-protocol. When she receives the message, she will run $Ver(check, \sigma_B)$, if it equals 1, she will accept the check. Otherwise, she will invoke the *recover* sub-protocol.

Main Protocol

A: $log(B, l, a, cipher, k)$
B: $log(A, l, check)$
$A \rightarrow B$: $f_{\mathbf{EOO}}, B, l, h(k), cipher, E_{TTP}(l, k), PS_A(a)$
B: **if not** $Ver(a, PS_A(a))$ **then** stop
else $log(A, l, h(k), cipher, E_{TTP}(l, k), PS_A(a))$
$B \rightarrow A$: $f_{\mathbf{EOR}}, A, l, PS_B(b)$
A: **if** times out **then abort**
elseif not $Ver(b, PS_B(b))$ **then abort**
else $log(B, l, PS_B(b))$
$A \rightarrow B$: $f_{\mathbf{NRO}}, B, l, k, FS_A(a)$
B: **if** times out **then** call **recover**[*X*:=*B*,*Y*:=*A*]
else $log(A, l, k, FS_A(a))$
$B \rightarrow A$: $f_{\mathbf{NRR}}, A, l, FS_B(b)$
A: **if** *A* times out **then** call **recover**[*X*:=*A*,*Y*:=*B*]

Recover Sub-protocol. Whenever necessary, Alice/Bob (noted by *X*) will invoke the *recover* protocol to let the TTP decide whether finish or abort the payment process.

Step 1, *X* sends to the TTP $E_{TTP}(k)$, $PS_A(a) = \sigma_A'$, *check*, $PS_B(check) = \sigma_B'$ to initiate a recover process. Because of the resilient channel between *X* and the TTP, this message will eventually arrives the TTP.

Step 2, when the TTP receive the message, it will first check whether the protocol has already been recovered or aborted, if so, it will stop because it is sure that both parties have got the recovered items or the abort confirmation. Then it will decrypt $E_{TTP}(k)$ with its secret key sk_{TTP}, if succeeds, it will run $PVer(a, \sigma_A')$ and $PVer(check, \sigma_B')$. If both equals 1, the TTP will run $Convert(a, \sigma_A')$ and $Convert(check, \sigma_B')$. After all these operations succeed, TTP will log the message and the variable *recovered*, then send the $FS_A(check) = \sigma_B$ to Alice and $FS_A(a) = \sigma_A$ & k to Bob. If either checking fails, it will abort the protocol and send confirmations to Alice and Bob.

Recover Sub-protocol

$X \rightarrow TTP$: $f_{\mathrm{RecX}}, Y, l, h(cipher), \mathrm{h}(k), E_{TTP}(k), PS_A(a), PS_B(b)$
TTP: $log(f_{\mathrm{RecX}}, A, B, l, h(cipher), \mathrm{h}(k), E_{TTP}(k), PS_A(a), PS_B(b))$
if $h(k) \neq h(D_{TTP}(E_{TTP}(k)))$ **or** *aborted* **or** *recovered* **then** stop
else if $PVer(a, PS_A(a)) \neq 1$ or $PVer(a, PS_A(a)) \neq 1$ **then** stop

else *recovered*=true
$Convert(PS_A(a), x_{2A})$ and $Convert(PS_B(b), x_{2B})$
$log(A, B, l, recovered, FS_A(a), k, FS_B(b))$
$TTP \rightarrow A$: $f_{\mathbf{NRR}}$, A, l, $FS_A(a)$
$TTP \rightarrow B$: $f_{\mathbf{NRO}}$, B, l, k, $FS_B(b)$

Inquiry Sub-protocol. After recovering from local system crashes, Alice/Bob (denoted as *X*) can invoke the *inquiry* sub-protocol to check the current status of the transaction and get what s/he deserves.

Step 1, *X* sends an inquiry request to the TTP. Because of the resilient channel between *X* and the TTP, this message will eventually arrives the TTP.
Step 2, on the inquiry request, TTP will check the current status of the protocol according to the label *l*. If no record is available, that means that protocol has not been submitted to TTP and *X* can directly recover the protocol run with *Y*. So TTP will just need to return a *null* message to X. If the protocol has been recovered, TTP will send the recovered message to *X*, that is, FS_A(a), *k* (for Bob) or $FS_A(check)$ (for Alice). If the protocol has been aborted, TTP will send the abort confirmation to *X*.

Inquiry Sub-protocol
$X \rightarrow TTP$: $f_{\mathrm{Inq}X}$, $\mathbf{Inq_X}$
TTP: **if** *aborted* **then**
$TTP \rightarrow X$: $f_{\mathbf{Con}a}$, A, B, l, $\mathbf{Con}_a$
TTP: **elseif** *recovered* **then**
if *X*=*A* then
$TTP \rightarrow A$: $f_{\mathbf{NRR}}$, A, l, $FS_A(a)$
else
$TTP \rightarrow B$: $f_{\mathbf{NRO}}$, B, l, k, $FS_B(b)$
else
$TTP \rightarrow X$: **null**

Abort Sub-protocol. In step 2 of the main protocol, Alice can invoke this sub-protocol to make the TTP abort this payment protocol run.

Step 1, Alice sends an abort request to the TTP. Because of the resilient channel between *X* and the TTP, this message will eventually arrives the TTP.
Step 2, if the protocol has not been recovered or aborted, the TTP will abort the protocol and log the message and the variable *aborted*, then send confirmations ($\mathbf{Con}_a$) to both parties.

Abort Sub-protocol
$X \rightarrow TTP$: $f_{\mathbf{Abort}}$, l, B, **abort**
TTP: **if** *aborted* **or** *recovered* **then** stop
else *aborted*=true
$log(A, B, l, aborted)$
$TTP \rightarrow A$: $f_{\mathbf{Con}a}$, A, B, l, $\mathbf{Con}_a$
$TTP \rightarrow B$: $f_{\mathbf{Con}a}$, A, B, l, $\mathbf{Con}_a$

4 Analysis of the Protocol

Following is the analysis with respect to the dependability definition in Section 2.

Claim 1. *Assuming the channel between Alice and Bob is unreliable and adopted cryptographic tools are secure, the protocol satisfies the effectiveness requirement.*

Proof: When both Alice and Bob are honest, thus they will follow the protocol to send messages. If the probability of successful transmission in the unreliable channel is δ, then the probability of successful execution of one main protocol run will roughly be δ^4. Even it's small, but it means successful execution without TTP's involvement is still possible. Thus the protocol satisfies the effectiveness requirement.

Claim 2. *Assuming the channels between the TTP and Alice/Bob are resilient, adopted cryptographic tools are secure and the TTP is honest, the protocol satisfies the fairness requirement.*

Proof: The fairness can be proved considering 3 aspects: fairness for Alice, fairness for Bob and recovered fairness after TTP crashes.

- *Fairness for Alice* Assuming Alice is honest, then risks she may faces include:

1) She did not receive any message or the message is invalid in step 3. She can request abort to prevent that Bob may call a recovery later. If Bob's recovery request arrives to the TTP before her abort request, the TTP still will send the recovered goods and evidence to her. Thus will not affect her benefit.
2) She did not receive any message or the message is invalid in step 5. She can submit a recovery request, because the TTP is honest, the exchange will be forced to complete. If Bob sent a recovery request during this period, the result will be the same; if Bob sent an abort request which arrived before Alice's recovery request, the exchange will be aborted by the TTP, and no party can gain advantage.
3) Local system crashes. After Alice recovers from local system crash, she can instantly invoke inquiry sub-protocol to check the current status; if she has submitted abort or recover request before her crash, she will get proper messages (abort confirmation or recovered messages) from TTP; if Bob has submitted recover request before or during her crash, she will get recovered messages from TTP; if no involvement before or during her crash, she can simply contact Bob to continue the transaction. So her fairness is assured.

- *Fairness for Bob* Assuming Bob is honest, then risks he may faces include:

1) He did not receive any message or the message is invalid in step 2. He can simply stop without any risk. And at this time, Alice cannot call recovery.
2) He did not receive any message or the message is invalid in step 4. He can request recovery and the exchange will be forced to complete. If Alice request recovery at the same time, the result will be the same.
3) Local system crashes. After Bob recovers from local system crash, he can instantly invoke inquiry sub-protocol to check the current status; if he has submitted recover request before his crash, he will get recovered messages from TTP;

if Alice has submitted abort or recover request before or during his crash, he will get proper messages (abort confirmation or recovered messages) from TTP; if no involvement before or during his crash, he can simply contact Alice to continue the transaction. So his fairness is assured.

- *Recovered fairness after TTP crashes* Cases of TTP crashes include:

1) Alice has submitted abort request before TTP crashes, and TTP has sent both parties the abort confirmation. Because TTP has logged request message and the variable *aborted*, so after TTP recovers the information about this protocol run, the TTP will deny any later recovery request by either Alice or Bob.
2) Alice/Bob has submitted recover request before TTP crashes, and TTP has sent both parties the recovered messages. Because TTP has logged the request message and the variable recovered, so after TTP recovers the information about this protocol run, the TTP can re-run the recovery operations (if necessary) and will ignore Alice's later abort request.
3) Alice/Bob has submitted abort/recover request during TTP crashes. Alice/Bob can re-submit request after TTP's recovery or TTP can actively broadcast the crashes information so that all requesting parties can re-submit their requests.

Claim 3. *Assuming the channels between the TTP and Alice/Bob are resilient, adopted cryptographic tools are secure and the TTP is honest, the protocol satisfies timeliness requirement.*

Proof: Alice can conclude the protocol in one of the two ways:

1) requesting abort before sending the message of step 3.
2) requesting recovery in any other time.

Bob can conclude the protocol in one of the three ways:

1) stopping at any time before sending the message of step 2.
2) requesting recovery in any other time.

With the channel assumption, the abort confirmation or the recovered information will arrive to both parties in a finite amount of time. And all these conclusions, as discussed in the proof of claim 2, will not hurt either party's interests. So the timeliness is guaranteed.

Claim 4. *Assuming the channels between the TTP and Alice/Bob are resilient, adopted cryptographic tools are secure, the TTP is honest, the protocol satisfies non-repudiation requirement.*

Proof: When the exchange succeeds, either by following the main protocol or recovered by the TTP (including recovered message after inquiry), Alice will get $FS_A(check)= \sigma_B$, and Bob will get $FS_A(a)= \sigma_A$ & k. So Alice can convince outside parties that Bob has received goods and claim her money from Bob's bank. Similarly, Bob can prove that Alice has sent goods.

Claim 5. *Assuming the channels between the TTP and Alice/Bob are resilient, adopted cryptographic tools are secure, the TTP is honest, the protocol guarantees transparent recoverability.*

Proof: Either the TTP is involved or not, the resulting message (FS_B(*check*), FS_A(a) and *k*) are just the same, so the protocol is transparent recoverable.

With all these claims, we can easily see that the protocol is dependable:

Theorem 1. *Assuming the channels between the TTP and Alice/Bob are resilient, adopted cryptographic tools are secure and the TTP is honest, the protocol is dependable.*

5 Conclusions

In this paper, we produce a dependable transaction protocol with transparent recoverability. We have shown that the protocol are practical as it has high recoverability and can survive relatively unreliable network. To be more precisely about effect of every factor in the protocol like network/system reliability, honesty of both parties and etc, we are building an agent-based platform for analysis and verification. Then we can see how dependable protocol can be applied in different environments.

References

1. N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In *Proceedings of the fourh ACM Conference on Computer and Communications Security*, 1997.
2. Y. Dodis, L. Reyzin. Breaking and repairing optimistic fair exchange from PODC 2003. In *Proceedings of the 2003 ACM workshop on Digital rights management*, 2003.
3. P. Liu, P. Ning, and S. Jajodia. Avoiding loss of fairness owing to process crashes in fair data exchange protocols. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks, Workshop on Dependability despite Malicious Faults*, 2000.
4. O. Markowitch, S. Kremer and D. Gollmann. On Fairness in Exchange Protocols. In *Proceedings of Information Security and Cryptology (ICISC 2002). LNCS 2587*, Springer-Verlag, 2002.
5. S. Micali. Certified e-mail with invisible post offices. Available from author: an invited presentation at the RSA'97 conference, 1997.
6. H. Pagnia and F. C. Gartner. On the impossibility of fair exchange without a trusted third party. *Tech. Rep. TUD-BS-1999-02 (March)*, Darmstadt University of Technology, 1999.
7. J. M. Park, E. K. P. Chong, H. J. Siegel. Constructing fair-exchange protocols for E-commerce via distributed computation of RSA signatures. *Proceedings of the twenty-second annual symposium on Principles of distributed computing*. 2003.
8. H. Wang and H. Guo. Fair Payment Protocols for E-Commerce. In *Proceedings of Fourth IFIP Conference on e-Commerce, e-Business, and e-Government (I3E'04)*. Building the E-Society: E-Commerce, E-Business and E-Government, Kluwer academic publishers, 2004.